A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing

Sally Floyd, Van Jacobson, Steven McCanne*
Lawrence Berkeley Laboratory, University of California, Berkeley, CA 94720
floyd, van, mccanne@ee.lbl.gov

Ching-Gung Liu†
University of Southern California, Los Angeles, CA 90089
charley@carlsbad.usc.edu

Lixia Zhang†
Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304
lixia@parc.xerox.com

Submitted to IEEE/ACM Transactions on Networking

November 13, 1995

Abstract

This paper[1] describes SRM (Scalable Reliable Multicast), a reliable multicast framework for application level framing and light-weight sessions. The algorithms of this framework are efficient, robust, and scale well to both very large networks and very large sessions. The framework has been prototyped in wb, a distributed whiteboard application, and has been extensively tested on a global scale with sessions ranging from a few to more than 1000 participants. The paper describes the principles that have guided our design, including the IP multicast group delivery model, an end-to-end, receiver-based model of reliability, and the application level framing protocol model. As with unicast communications, the performance of a reliable multicast delivery algorithm depends on the underlying topology and operational environment. We investigate that dependence via analysis and simulation, and demonstrate an adaptive algorithm that uses the results of previous loss recovery events to adapt the control parameters used for future loss recovery. With the adaptive algorithm, our reliable multicast delivery algorithm provides good performance over a wide range of underlying topologies.

*Supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

†Supported in part by the Advanced Research Projects Agency, monitored by Fort Huachuca under contract DABT63-94-C-0073.

[1] An earlier version of this paper appeared in ACM SIGCOMM 95. This version corrects errors in the graphs of that earlier version.

1 Introduction

Several researchers have proposed generic reliable multicast protocols, much as TCP is a generic transport protocol for reliable unicast transmission. In this paper we take a different view: unlike the unicast case where requirements for reliable, sequenced data delivery are fairly general, different multicast applications have widely different requirements for reliability. For example, some applications require that delivery obey a total ordering while many others do not. Some applications have many or all the members sending data while others have only one data source. Some applications have replicated data, for example in an n-redundant file store, so several members are capable of transmitting a data item while for others all data originates at a single source. These differences all affect the design of a reliable multicast protocol. Although one could design a protocol for the worst-case requirements, e.g., guaranteeing totally ordered delivery of replicated data from a large number of sources, such an approach results in substantial overhead for applications with more modest requirements. One cannot make a single reliable multicast delivery scheme that simultaneously meets the functionality, scalability, and efficiency requirements of all applications.

The weakness of "one size fits all" protocols has long been recognized. In 1990 Clark and Tennenhouse proposed a new protocol model called Application Level Framing (ALF) which explicitly includes an application's semantics in the design of that application's protocol [CT90]. ALF was later elaborated with a light-weight rendezvous mechanism based on the IP multicast distribution model, and with a notion of receiver-based adaptation for unreliable, real-time applications such as audio and video conferencing. The result, known as Light-Weight Sessions (LWS), has been very successful in the design of wide-area, large-scale, conferencing applications. This paper further evolves the principles of ALF and LWS to add a framework for scalable reliable multicast (SRM).

ALF says that the best way to meet diverse application requirements is to leave as much functionality and flexibility as possible to the application. Therefore our algorithms are designed to meet only the minimal definition of reliable multicast, i.e., eventual delivery of all the data to all the group members, without enforcing any particular delivery order. We believe that if the need arises, machinery to enforce a particular delivery order can be easily added on top of this reliable delivery service.

The design is also heavily based on the group delivery model that is the centerpiece of the IP multicast protocol [D91]. In IP multicast, data sources simply send to the group's multicast address (a normal IP address chosen from a reserved range of addresses) without needing any advance knowledge of the group membership. To receive any data sent to the group, receivers simply announce that they are interested (via a "join" message multicast on the local subnet) — no knowledge of the group membership or active senders is required. Each receiver joins and leaves the group individually, without affecting the data transmission to any other member. Our multicast delivery framework further enhances the multicast group concept by maximizing information and data sharing among all the members, and strengthens the individuality of membership by making each member responsible for its own correct reception of all the data.

Finally, our design attempts to follow the core design principles of TCP/IP. First, we require only the basic IP delivery model — best-effort with possible duplication and reordering of packets — and build the reliability on an end-to-end basis. No change or special support is required from the underlying IP network. Second, in a fashion similar to TCP adaptively setting timers or congestion control windows, our algorithms dynamically adjust their control parameters based on the observed performance within a session. This allows applications using this model to adapt to a wide range of group sizes, topologies and link bandwidths while maintaining robust and high performance.

The paper proceeds as follows: Section 2 discusses general issues for reliable multicast delivery. Section 3 describes in detail the reliable multicast algorithm embedded in the wb implementation. Section 4 discusses the performance of the algorithm in simple topologies such as chains, stars, and bounded-degree trees, and Section 5 presents simulation results from more complex topologies. Section 6 examines the behavior of the loss recovery algorithms as a function of the timer parameters. Section 7 discusses extensions to the basic scheme embedded in wb, such as adaptive algorithms for adjusting the timer parameters and algorithms for local recovery. Section 8 discusses both the application-specific aspects of wb's reliable multicast algorithms as well as the aspects of the underlying approach that have general applicability. Section 9 discusses related work on reliable multicast. Section 10 discusses future work on the congestion control algorithms. Finally, Section 11 presents conclusions.

2 The design of reliable multicast

2.1 Reliable data delivery: adding the word "multicast"

The problem of reliable unicast data delivery is well understood and a variety of well-tested solutions are available. However, adding the word "multicast" to the problem statement significantly changes the solution set. For example, in any reliable protocol some party must take responsibility for loss detection and recovery. Because of the "fate-sharing" implicit in unicast communication, i.e., the data transmission fails if either of the two ends fails, either the sender or receiver can take on this role. In TCP, the sender times transmissions and keeps retransmitting until an acknowledgment is received. NETBLT [CLZ87] uses the opposite model and makes the receiver responsible for all loss detection and recovery. Both approaches have been shown to work well for unicast.

However, if a TCP-style, sender-based approach is applied to multicast distribution, a number of problems occur. First, because data packets trigger acknowledgments (positive or negative) from all the receivers, the sender is subject to the well-known ACK implosion effect. Also, if the sender is responsible for reliable delivery, it must continuously track the changing set of active receivers and the reception state of each. Since the IP multicast model deliberately imposes a level of indirection between senders and receivers (i.e., data is sent to the multicast group, not to the set of receivers), the receiver set may be expensive or impossible to obtain. Finally, the algorithms that are used to adapt to changing network conditions tend to lose their meaning in the case of multicast. E.g., how should the round-trip time estimate for a retransmit timer be computed when there may be several orders of magnitude difference in propagation time to different receivers? What is a congestion window if the delay-bandwidth product to different receivers varies by orders of magnitude? What self-clocking information exists in the ACK stream(s) if some receivers share one bottleneck link and some another?

These problems illustrate that single-point, sender-based control does not adapt or scale well for multicast delivery. Since members of a multicast group have different communication paths and may come and go at any time, the "fate-shared" coupling of sender and receiver in unicast transmissions does not generalize to multicast. Thus it is clear that receiver-based reliability is a far better building block for reliable multicast [PTK94].

Another unicast convention that migrates poorly to multicast has to do with the vocabulary used by the sender and receiver(s) to describe the progress of their communication. A receiver can request a retransmission either in application data units ("sector 5 of file sigcomm-slides.ps") or in terms of the shared communication state ("sequence numbers 2560 to 3071 of this conversation"). Both models have been used successfully (e.g., NFS uses the former and TCP the latter) but, because the use of communication state for naming data allows the protocol to be entirely independent of any application's namespace, it is by far the most popular approach for unicast applications. However, since multicast transmission tends to have much weaker and more diverse state synchronization than does unicast, using shared communication state to name data does not work well in the multicast case.

For example, if a receiver joins a conversation late and receives sequence numbers 2560 to 3071, it has no idea of what's been missed (since the sender's starting number is arbitrary) and so can neither do anything useful with the data nor make an intelligent request for retransmission. If receivers hear from a sender again after a lengthy network partition, they have no way of knowing whether "2560" is a retransmission of data they received before the partition or is completely new (due to sequence number wrapping during the partition). Thus the "naming in application data units (ADUs)" model works far better for multicast.

Use of this model also has two beneficial side effects. As [CT90] points out, a separate protocol namespace can impose delays and inefficiencies on an application, e.g., TCP will only deliver data in sequence even though a file transfer application might be perfectly happy to receive sectors in any order. The ADU model eliminates this delay and puts the application back in control. Also, since ADU names can be made independent of the sending host, it is possible to use the anonymity of IP multicast to exploit the redundancy of multiple receivers. E.g., if some receiver asks for a retransmit of "sigcomm-slides.ps sector 5", any member who has a copy of the data, not just the original sender, can carry out the retransmission.

2.2 Reliable multicast requirements

While the ALF model says that applications should be actively involved in their communications and that communication should be done in terms of ADUs rather than some generic protocol namespace, we do not claim that every application's protocol must be completely different from every other's or that there can be no shared design or code. A great deal of design commonality is imposed simply because different applications are attempting to solve the same problem: scalable, reliable, multipoint communication over the Internet. As Section 2.1 pointed out, just going from unicast to multicast greatly limits the viable protocol design choices. In addition, experience with the Internet has shown that successful protocols must accommodate many orders of magnitude variation in every possible dimension. While several algorithms meet the constraints of Section 2.1, very few of them continue to work if the delay, bandwidth and user population are all varied by factors of 1000 or more.

In the end we believe the ALF model results in a skeleton or template which is then filled in with application specific details. Portions of that skeleton are completely determined by network dynamics and scaling considerations and apply to any application. So, for example, the scalable request and repair algorithms described in Sections 3 through 7 are completely generic and apply to a wide variety of reliable multicast applications. Each different application supplies this reliability framework with a namespace to talk about what data has been sent and received; a policy and machinery to determine how bandwidth should be apportioned between a participant in the group, the group as a whole, and other users of the net; and a local send policy that a participant uses to arbitrate the different demands on its bandwidth (e.g., locally originated data, repair requests and responses, etc.). It is the intent of this paper to describe the skeleton common to scalable, reliable multicast applications. However, to make the ideas concrete, we first describe a complete, widely used application — wb, the LBNL network whiteboard — that has been implemented according to this model. After mentioning some details of its operation that are direct implications of the design considerations in Section 2.1, we then factor out the wb specifics to expose the generic, scalable, reliable multicast skeleton underneath. The remaining sections of this paper are an exploration of that skeleton.

2.3 The wb framework

Wb is a network conferencing tool designed and implemented by McCanne and Jacobson [J92, J94a, M92] that provides a distributed whiteboard. The whiteboard separates the drawing into pages, where a new page can correspond to a new viewgraph in a talk or the clearing of the screen by a member of a meeting. Any member can create a page and any member can draw on any page.[2] Each member is identified by a globally unique identifier, the Source-ID, and each page is identified by the Source-ID of the initiator of the page and a page number locally unique to that initiator. Each member drawing on the whiteboard produces a stream of drawing

[2]There are floor control mechanisms, largely external to wb, that can be used if necessary to control who can create or draw on pages. These can be combined with normal Internet privacy mechanisms (e.g., symmetric-key encryption of all the wb data) to limit participation to a particular group and/or with normal authentication mechanisms (e.g., participants signing their drawing operations via public-key encryption of a cryptographic hash over the data). The privacy, authentication and control mechanisms are completely orthogonal to the reliability machinery that is the subject of this paper and will not be described here. For further details see [MJ95, J94].

operations, or "drawops", that are timestamped and assigned sequence numbers, relative to the sender. Most drawing operations are idempotent and are rendered immediately upon receipt. Each member's graphics stream is independent from that of other sites.

The following assumptions are made in wb's reliable multicast design:

- All data has a unique, persistent name.

  This global name consists of the end host's Source-ID and a locally-unique sequence number.

- The name always refers to the same data.

  It is impossible to achieve consistency among different receivers in the face of late arrivals and network partitions if, say, drawop "floyd:5" initially means a blue line and later turns into a red circle. This does not mean that the drawing can't change, only that drawops must effect the change. E.g., to change a blue line to a red circle, a "delete" drawop for "floyd:5" is sent, then a drawop for the circle is sent.

- Source-ID's are persistent.

  A user will often quit a session and later re-join, obtaining the session's history from the network. By ensuring that Source-ID's are persistent across invocations of the application, the user maintains ownership of any data created before quitting.

- IP multicast datagram delivery is available.

- All participants join the same multicast group; there is no distinction between senders and receivers.

Wb has no requirement for ordered delivery because most operations are idempotent. Operations that are not strictly idempotent, such as a "delete" that references an earlier drawop, can be patched after the fact, when the missing data arrives. A receiver uses the timestamps on the drawing operations to determine the rendering order. This coarse synchronization mechanism captures the temporal causality of drawing operations at a level appropriate for the application, without the added complexity and delay of protocols that provide guaranteed causal ordering.

3 Wb's instantiation of the reliable multicast algorithm

Whenever new data is generated by wb, it is multicast to the group. Each member of the group is individually responsible for detecting loss and requesting retransmission. Loss is normally detected by finding a gap in the sequence space. However, since it is possible that the last drawop of a set is dropped, each member sends low-rate, periodic, session messages that announce the highest sequence number received from every member that has written on the page currently being displayed. In addition to the reception state, the session messages contain timestamps that are used to estimate the distance (in time) from each member to every other (described in Section 3.1).

When receiver(s) detect missing data, they wait for a random time determined by their distance from the original source of the data, then send a repair request (the timer calculations are described in detail in Section 3.2). As with the original data, repair requests and retransmissions are always multicast to the whole group. Thus, although a number of hosts may all miss the same packet, a host close to the point of failure is likely to timeout first and multicast the request. Other hosts that are also missing the data hear that request and suppress their own request. (This prevents a request implosion.) Any host that has a copy of the requested data can answer a request. It will set a repair timer to a random value depending on its distance from the sender of the request message and multicast the repair when the timer goes off. Other hosts that had the data and scheduled repairs will cancel their repair timers when they hear the multicast from the first host. (This prevents a response implosion). In a topology with diverse transmission delays, a lost packet is likely to trigger only a single request from a host just downstream of the point of failure and a single repair from a host just upstream of the point of failure. Section 5 explores in more detail the number of requests and repairs in different topologies.

3.1 Session messages

As mentioned above, each member sends periodic session messages that report the sequence number state for active sources. Receivers use these session messages to determine the current participants of the session and to detect losses. The average bandwidth consumed by session messages is limited to a small fraction (e.g., 5%) of the session data bandwidth using the algorithm developed for vat and described in [SCFJ94].

In a large, long-lived session, the state would become unmanageable if each receiver had to report the sequence numbers of everyone who had ever written to the whiteboard. The "pages" mentioned above are used to partition the state and prevent this explosion. Each member only reports the state of the page it is currently viewing. If a receiver joins late, it may issue *page requests* to learn the existence of pages and the sequence number state in each page. We omit the details of the page state recovery protocol as it is almost identical to the repair request / response protocol for data.

In addition to state exchange, receivers use the session messages to estimate the one-way distance between nodes. All whiteboard packets, including session packets, include a Source-ID and a timestamp. The session packet timestamps are used to estimate the host-to-host distances needed by the repair algorithm.

The timestamps are used in a highly simplified version of the NTP time synchronization algorithm [M84]. Assume that host A sends a session packet P_1 at time t_1 and host B receives P_1 at time t_2. At some later time, t_3, host B generates a session packet P_2, marked with (t_1, Δ) where Δ=$t_3 - t_2$ (time t_1 is included in P_2 to make the algorithm robust to lost session packets). Upon receiving P_2 at time t_4, host A can estimate the latency from host B to host A as $(t_4 - t_1 - \Delta)/2$. Note that while this estimate does assume that the paths are symmetric, it does not assume synchronized clocks.

3.2 Loss recovery

The loss recovery algorithm provides the foundation for reliable delivery. In this section we describe the loss recovery algorithm originally designed for wb; Section 7.1 describes a modified version of this algorithm with an adaptive adjustment of the timer parameters.

When host A detects a loss, it schedules a repair request for a random time in the future. The request timer is chosen from the uniform distribution on $[C_1 d_{S,A}, (C_1 + C_2)\, d_{S,A}]$ seconds, where $d_{S,A}$ is host A's estimate of the one-way delay to the original source S of the missing data, and C_1 and C_2 are parameters of the request algorithm. When the request timer expires, host A multicasts a request for the missing data, and doubles the request timer to wait for the repair.

If host A receives a request for the missing data before its own request timer for that data expires, then host A does a (random) exponential backoff, and resets its request timer.[3] That is, if the current timer had been chosen from the uniform distribution on

$$2^i\,[C_1 d_{S,A}, (C_1 + C_2)\, d_{S,A}]\,,$$

then the backed-off timer is randomly chosen from the uniform distribution on

$$2^{i+1}\,[C_1 d_{S,A}, (C_1 + C_2)\, d_{S,A}]\,.$$

When host B receives a request from A that host B is capable of answering, host B sets a repair timer to a value from the uniform distribution on

$$[D_1 d_{A,B}, (D_1 + D_2)\, d_{A,B}]$$

seconds, where $d_{A,B}$ is host B's estimate of the one-way delay to host A, and D_1 and D_2 are parameters of the repair algorithm. If host B receives a repair for the missing data before its repair timer expires, then host B cancels its repair timer. Otherwise, when host B's repair timer expires host B multicasts the repair. Because host B is not responsible for host A's reliable data reception, it does not verify whether host A actually receives the repair.

Due to the probabilistic nature of these algorithms, it is not unusual for a dropped packet to be followed by more than one request. Thus, a host could receive a duplicate request immediately after sending a repair, or immediately after receiving a repair in response to its own earlier request. In order to prevent duplicate requests from triggering a responding set of duplicate repairs, host B ignores requests for data D for $3\, d_{S,B}$ seconds after sending or receiving a repair for that data, where host S is either the original source of data D or the source of the first request.

Because data represents idempotent operations, loss recovery can proceed independently from the transmission of new data. Similarly, recovery for losses from two different sources can also proceed independently. Since transmission bandwidth is often limited, a single transmission rate is allocated to control the throughput across all these different modes of operation, while the application determines the order of packet transmission according to their relative importance. In wb, the highest priority packets are repairs for the current page, middle priority are new data, and lowest priority are repairs for previous pages.

3.3 Bandwidth limitations

The congestion control mechanism for whiteboard sessions is based on a (fixed, in current implementations) maximum bandwidth allocation for each session. Each wb session has a sender bandwidth limit advertised as part of the sd announcement. A typical value is 64 Kbps; in this case a wb session costs no more (and typically considerably less) than the accompanying audio session. Individual members use a token bucket rate limiter to enforce this peak rate on transmissions. This peak rate is mostly relevant when a source distributes a postscript file for a new page of the whiteboard, or when a late arrival requests the past history of the whiteboard session.

3.4 Recovery from partitioning

The whiteboard does not require special mechanisms for the detection or recovery from network partitioning. Because wb relies on the underlying concept of an IP multicast group, where members can arrive and depart independently, wb does not distinguish a partitioning from a normal departure of members from the wb session.

During a partition of a session, users can simply continue using the whiteboard in the connected components of the

[3] In the simulations described in later sections, we don't backoff the request timer for every duplicate request that is received. For example, if a member receives several duplicate requests immediately after receiving the initial request, then that member only backs off its request timer once, not several times. After the initial timer backoff, we only backoff the timer again if a request is received close to the time when the timer is set to expire. More precisely, when we backoff the request timer, then we set an *ignore-backoff* variable to a time halfway between the current time and the expiration time, and we ignore additional duplicate requests until *ignore-backoff* time.

partitions. Because pages are identified by the Source-ID of the initiator of the page, along with the page number for that initiator, members can continue creating new pages during the partition (e.g., "Floyd:3" in one half of the partition, and "Zhang:5" in the other). After recovery each page will still have a unique page ID and the repair mechanism will distribute any new state throughout the entire group.

Almost all of the design described in this section is present in version 1.59 of wb; some omissions are pending implementation. These omissions include the measurements of one-way delays and the rate-limiting mechanisms.

4 Request/repair algorithms for simple topologies

Building on our initial design experiences in wb, we turn to a more general investigation of the loss recovery algorithms. The algorithms described in the remainder of the paper have been implemented only within our simulation framework.

Given that multiple hosts may detect the same losses, and multiple hosts may attempt to handle the same repair request, the goal of the request/repair timer algorithms is to de-synchronize host actions to keep the number of duplicates low. Among hosts that have diverse delays to other hosts in the same group, this difference in delay can be used to differentiate hosts; for hosts that have similar delays to reach others, we can only rely on randomization to de-synchronize their actions.

This section discusses a few simple, yet representative, topologies, namely chain, star, and tree topologies, to provide a foundation for understanding the loss recovery algorithms in more complex environments. For a chain the essential feature of a loss recovery algorithm is that the timer value be a function of distance. For a star topology the essential feature of the loss recovery algorithm is the randomization used to reduce implosion. Request/repair algorithms in a tree combine both the randomization and the setting of the timer as a function of distance. This section shows that the performance of the loss recovery algorithms depends on the underlying network topology.

4.1 Chains

Figure 1 shows a chain topology where all nodes in the chain are members of the multicast session. Each node in the underlying multicast tree has degree at most two. The chain is an extreme topology where a simple deterministic loss recovery algorithm suffices; in this section we assume that C_1, D_1=1, and that C_2, D_2=0.

For the chain, as in most of the other scenarios in this paper, link distance and delay are both normalized. We assume that packets take one unit of time to travel each link, i.e. all links have distance of 1.

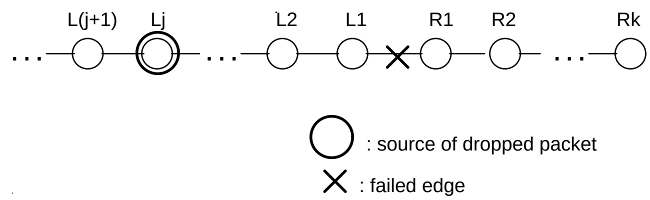

Figure 1: A chain topology.

In Figure 1 the nodes in the chain are labeled as either to the right or to the left of the congested link. Assume that source L_j multicasts a packet that is subsequently dropped on link (L_1, R_1), and that the second packet sent from source L_j is not dropped. We call the edge that dropped the packet, whether due to congestion or to other problems, the *congested* link. Assume that the right-hand nodes each detect the failure when they receive the second packet from L_j.

Assume that node R_1 first detects the loss at time t, and that each link has distance 1. Then node R_1 multicasts a request at time $t + j$. Node L_1 receives the request at time $t + j + 1$ and multicasts a repair at time $t + j + 2$. Node R_k receives the repair at time $t + k + j + 2$.

Note that all nodes to the right of node R_1 receive the request from R_1 before their own request timers expire. We call this *deterministic suppression*. The reader can verify that, due to deterministic suppression, there will be only one request and one repair.

Had the loss repair been done by unicast, i.e. node R_k sent a unicast request to the source L_j as soon as it detected the failure and L_j sent a unicast repair to R_k as soon as it received the request, node R_k would not receive the repair until time $t + 2j + 3k$. Thus, with a chain and with a simple deterministic loss recovery algorithm, the furthest node receives the repair sooner than it would if it had to rely on its own unicast communication with the original source. While the original source and the intended recipient of the dropped packet could be arbitrarily far from the congested link, in the multicast repair algorithm both the request and the repair come from the node immediately adjacent to the congested link.

4.2 Stars

For the star topology in Figure 2 we assume that all links are identical and that the center node is not a member of the multicast group. For a star topology, setting the request timer as a function of the distance from the source is not an essential feature, as all nodes detect a loss at exactly the same time. Instead, the essential feature of the loss recovery algorithm in a star is the randomization used to reduce implosion; we call this *probabilistic suppression*.

For the star topology in Figure 2 assume that the first packet from node N_1 is dropped on the adjacent link. There are G members of the multicast session, and the other members

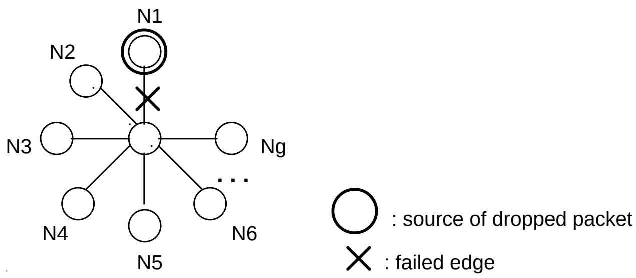

Figure 2: A star topology.

detect the loss at exactly the same time. For the discussion of this topology we assume that C_1, D_1=0; because all nodes detect losses and receive requests at the same time, C_1 and D_1 are not needed to amplify differences in delay. The only benefit in setting C_1 greater than 0 is to avoid unnecessary requests from out-of-order packets.

If C_2 is at most 1, then there will always be $G - 1$ requests. Increasing C_2 reduces the expected number of requests but increases the expected time until the first request is sent. For $C_2 > 1$, the expected number of requests is roughly $1 + (G - 2)/C_2$, and the expected delay until the first timer expires is $2C_2/G$ seconds (where one unit of time is one second).[4] For example, if C_2 is set to $\sqrt{G}$, then the expected number of requests is roughly $\sqrt{G}$, and the expected delay until the first timer expires is $2/\sqrt{G}$ seconds. The same remarks apply to D_2 with respect to repairs.

4.3 Bounded-degree trees

The loss recovery performance in a tree topology uses both the deterministic suppression described for chain topologies and the probabilistic suppression described for star topologies. Consider a network topology of a bounded-degree tree with N nodes where interior nodes have degree p. A tree topology combines aspects of both chains and stars. The timer value should be a function of distance, to enable requests and repairs to suppress request and repair timers at nodes further down in the tree. In addition, randomization is needed to reduce request/repair implosion from nodes that are at an equal distance from the source (of the dropped packet, or of the first request).

We assume that node S in the tree is the source of the dropped packet, and that link (B,A) drops a packet from source S. We call nodes on the source's side of the congested link (including node B) *good* nodes, and nodes on the other side of the congested link (including node A) *bad* nodes. Node A detects the dropped packet at time t, when it receives the next packet from node S. We designate node A as a *level-0* node, and we call a bad node a *level-i* node if it is at distance i from node A.

Assume that the source of the dropped packet is at distance j from node A. Node A's request timer expires at time

$$t + C_1 j + U_1[C_2]\, j,$$

where $U[C_2]$ denotes a uniform random variable between 0 and C_2. Assuming that node A's request is not suppressed, a level-i node receives node A's request at time

$$t + i + C_1 j + U_1[C_2]\, j.$$

Node B receives node A's repair request at time

$$t + 1 + C_1 j + U_1[C_2]\, j.$$

A bad level-i node detects the loss at time $t + i$, and such a node's request timer expires at some time

$$t + i + C_1(i + j) + U_2[C_2]\,(i + j).$$

Note that regardless of the values of $U_1[C_2]$ and $U_2[C_2]$, a level-i node receives node A's request by time $t + i + C_1 j + C_2 j$, and a level-i node's request timer expires no sooner than $t + i + C_1(i + j)$. If

$$t + i + C_1 j + C_2 j \le t + i + C_1(i + j),$$

that is, if

$$\frac{C_2}{C_1} j \le i,$$

then the level-i node's request timer will always be suppressed by the request from the level-0 node. Thus, the smaller the ratio C_2/C_1, the fewer the number of levels that could be involved in duplicate requests. This relation also demonstrates why the number of duplicate requests or repairs is smaller when the source (of the dropped packet, or of the request) is close to the congested link.

Note that the parameter C_1 serves two different functions. A smaller value for C_1 gives a smaller delay for node B to receive the first request. At the same time, for nodes further away from the congested link, a larger value for C_1 contributes to suppressing additional levels of request timers. A similar tradeoff occurs with the parameter C_2. A smaller value for C_2 gives a smaller delay for node B to receive the first repair request. At the same time, for topologies such as star topologies, a larger value for C_2 helps to prevent duplicate requests from session members at the same distance from the congested link. Similar remarks apply to the functions of D_1 and D_2 in the repair timer algorithm.

5 Simulations of the request and repair algorithms

For a given underlying network, set of session members, session sources, and congested link, it should be feasible to

[4] The $G - 1$ nodes all detect the failure at the same time, and all set their timers to a uniform value in an interval of width $2C_2$. If the first timer expires at time t, then the other $G - 2$ receivers receive that first request at time $t + 2$. So the expected number of duplicate requests is equal to the expected number of timers that expire in the interval $[t, t + 2]$.

analyze the behavior of the repair and request algorithms, given fixed timer parameters C_1, C_2, D_1, and D_2. However, we are interested in the repair and request algorithms across a wide range of topologies and scenarios. We use simulations to examine the performance of the loss recovery algorithms for individual packet drops in random and bounded-degree trees. We do not claim to be presenting realistic topologies or typical patterns of packet loss.

The simulations in this section show that the loss recovery algorithms with fixed timer parameters perform well in a random or bounded-degree tree when every node in the underlying tree is a member of the multicast session. The loss recovery algorithms perform somewhat less well for a sparse session, where the session size is small relative to the size of the underlying network and the members are scattered throughout the net. This motivates the development on the adaptive loss recovery algorithm in Section 7.1, where the timer parameters C_1, C_2, D_1, and D_2 are adjusted in response to past performance.

In these simulations the fixed timer parameters are set as follows: C_1, C_2=2, and D_1, D_2=$\log_{10} G$, where G is the number of members in the same multicast session. The choice of $\log_{10} G$ for D_1 and D_2 is not critical, but gives slightly better performance than D_1, D_2=1 for large G.

Each simulation constructs either a random tree or a bounded degree tree with N nodes as the network topology. Next, G of the N nodes are randomly chosen to be session members, and a source S is randomly chosen from the G session members.

We assume that messages are multicast to members of the multicast group along a shortest-path tree from the source of the message. In each simulation we randomly choose a link L on the shortest-path tree from source S to the G members of the multicast group. We assume that the first packet from source S is dropped by link L, and that receivers detect this loss when they receive the subsequent packet from source S.

5.1 Illustrating the simulator

In this section we show one of the tools that we use to verify that our simulator is implementing the loss recovery algorithms correctly. This figure also serves as a concrete illustration of the loss recovery algorithms in operation.

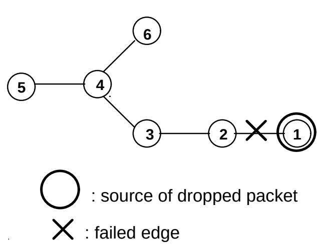

Figure 3: A simulation network for the figure above.

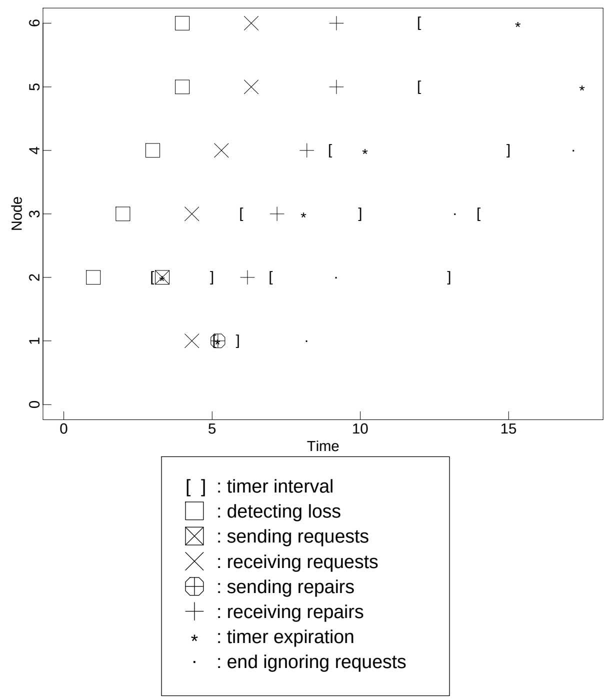

Figure 4: A request/repair exchange from a single dropped packet.

Figure 4 shows a single request/repair exchange for the network in Figure 3. This is one of a series of automated tests that we run after each change we make to our simulator. The underlying network shown in Figure 3 consists of a randomly-created tree of six nodes. A packet takes one unit of time to traverse each link.

In Figure 4 the x-axis shows time. The y-axis shows a row for each session member, indicating when timers are set and repair and request packets sent and received by that member. This simulation uses the fixed timer parameters C_1, C_2=2 and D_1, D_2=1.

For each member affected by the loss, we define the *loss recovery delay* as the time from when the member first detected the loss until the member first received a repair. The graph shows this loss recovery delay as a multiple of RTT, the roundtrip time from the receiver to the original source of the dropped packet. The simulator's summary statistics correctly report that the delay/RTT for the last node to receive the repair is 0.65. This is for node 1, which detects the loss at time 4, receives the repair at time 9.2, and has a RTT of 8 to the source of the dropped packet.

Note that with unicast communications the ratio of loss recovery delay to RTT is at least one. For a unicast receiver that detects a packet loss by waiting for a retransmit timer to time out, the typical ratio of delay to RTT is closer to 2. As the earlier discussion of chain topologies shows, with multicast loss recovery algorithms the ratio of delay to RTT can sometimes be less than one, because the request and/or repair could each come from a node close to the point of failure.

Figure 4 can be read in two ways to verify the correctness of the algorithms implemented in the simulator. First, a single row shows the history of a single member. We leave the verification of each row as an exercise for the reader. Second, for each multicast request or repair, the figure shows when that message was received by each of the other nodes.

5.2 Simulations on random trees

We first consider simulations on random labeled trees of N nodes, constructed according to the labeling algorithm in [Pa85, p.99]. These trees have unbounded degree, but for large N, the probability that a particular vertex in a random labeled tree has degree at most four approaches (approximately) 0.98 [Pa85, p.114]. Figure 5 shows simulations of the loss recovery algorithm for this case, where all N nodes in the tree are members of the multicast session (that is, G=N). For each graph the x-axis shows the session size G; twenty simulations were run for each value of G. Each simulation is represented by a jittered dot, and the median from the twenty simulations is shown by a solid line. The two dotted lines mark the upper and lower quartiles; thus, the results from half of the simulations lie between the two dotted lines.

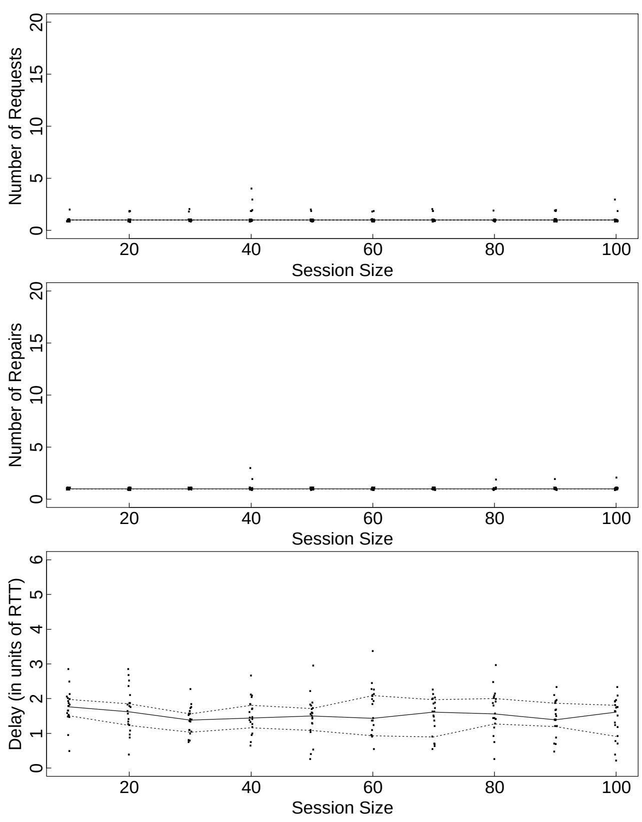

Figure 5: Random trees with a random congested link, where all nodes are members of the multicast session.

The top two graphs in Figure 5 show the number of requests and repairs to recover from a single loss. The bottom graph shows the loss recovery delay of the last node in the multicast session to receive the repair.

Figure 5 shows that the repair/request algorithm works well for a tree topology where all nodes of the tree are members of the multicast session. There is usually only one request and one repair. (Some lack of symmetry results from the fact that the original source of the dropped packet might be far from the point of failure, while the first request comes from a node close to the point of failure.) The average recovery delay for the farthest node is less than 2 RTT, competitive with the average delay available from a unicast algorithm such as TCP. The results are similar in simulations where the congested link is chosen adjacent to the source of the dropped packet, and for simulations on a bounded-degree tree of size N=G where interior nodes have degree 4.

5.3 Simulations on large bounded-degree trees

The performance of the loss recovery algorithms with fixed timer parameters is less optimal when the underlying network is a large bounded-degree tree. The underlying topology for the simulations in this section is a balanced bounded-degree tree of N=1000 nodes, with interior nodes of degree four. In these simulations the session size G is significantly less than N. For a session that is sparse relative to the underlying network, the nodes close to the congested link might not be members of the session.

As Figure 6 shows, the average number of repairs for each loss is somewhat high. In simulations shown in [FJLMZ95] where the congested link is always adjacent to the source, the number of repairs is low but the average number of requests is high.

[FJLMZ95] shows the performance of the loss recovery algorithm on a range of topologies. These include topologies where each of the N nodes in the underlying network is a router with an adjacent Ethernet with 5 workstations, point-to-point topologies where the edges have a range of propagation delays, and topologies where the underlying network is more dense that a tree. None of these variations that we have explored have significantly affected the performance of the loss recovery algorithms with fixed timer parameters.

6 Exploring the parameter space

As the previous section showed, a particular set of values for the timer parameters C_1, C_2, D_1, and D_2 that performs well in one scenario might not perform well in another scenario. In this section we choose a few simple topologies, and explore the behavior of the request/repair algorithms as a function of the request timer parameter C_2; C_1 is set to

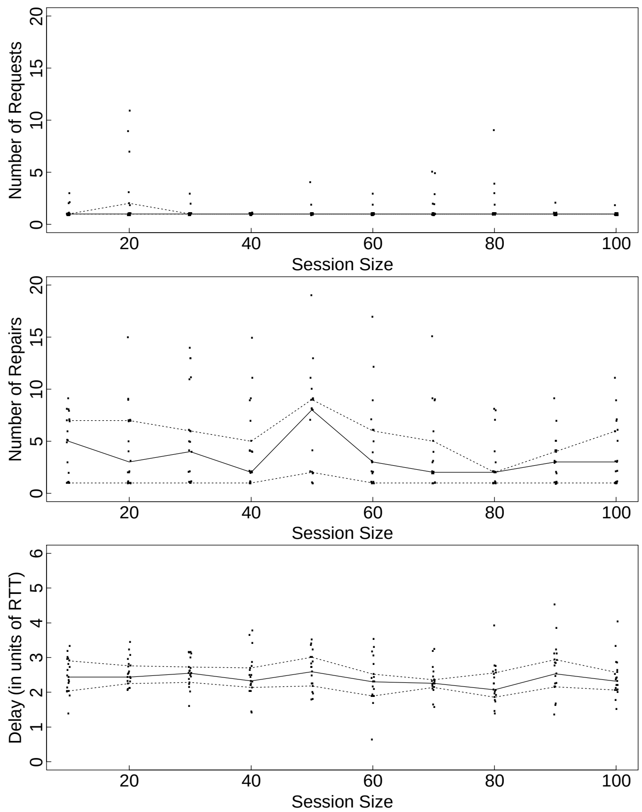

Figure 6: Bounded-degree tree, degree 4, 1000 nodes, with a random congested link.

2. In the following section we discuss adaptive algorithms where the timer parameters are adjusted as a function of the past performance of the loss recovery algorithms.

For a star topology with sparse trees, there is a clear tradeoff between the delay and the number of duplicates. In contrast, with a chain topology, setting C_2 to zero gives the optimal behavior both in terms of delay and in the number of duplicates. Although the performance for a dense tree is more complex than either a star or a chain, a small value for C_2 gives good performance in terms of both delay and duplicates.

Figure 7 shows the tradeoffs between delay and duplicates in a star topology of size 100. We define the *request delay* for a session member as the delay from when the request timer is set until a request was either sent by that member or received from another member. The top graph in Figure 7 contains a dot for each integer value of C_2 from 1 to 100, for the star topology described in Section 4.2. For each dot, the x-coordinate is the expected request delay for that value of C_2, and the y-coordinate is the expected number of requests.

More precisely, the x-coordinate is given by the expected request delay for the bad member closest to the source of the dropped packet, expressed as a multiple of the roundtrip time from that member to the source of the dropped packet. When there is not a unique bad member at the minimum distance

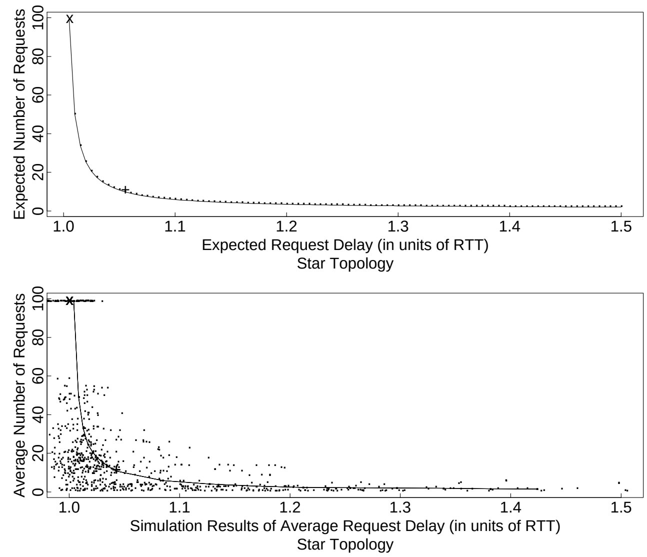

Figure 7: Tradeoff between delay and duplicates in a star topology.

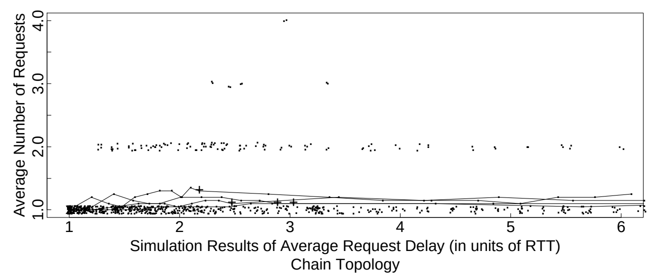

Figure 8: Tradeoff between delay and duplicates in a chain topology.

from the source, as in a star topology, then the x-axis shows the expected smallest request delay from those members at the minimum distance from the source. For a star topology this is the request delay for that member whose request timer expires first.

From the heuristic analysis in Section 4.2, the expected request delay (in units of the RTT of $2D$) is as follows:

$$\frac{C_1 D + C_2 D/G}{2D}$$
$$= C_1/2 + C_2/(2G) \ ,$$

where D is the distance in seconds from the source to a session member. From Section 4.2, the expected number of requests is estimated as $1 + (G-2)/C_2$. The "x" in Figure 7 shows the results for C_2=0, and the "+" shows the results for C_2=10. Thus the top graph of Figure 7 shows that increasing C_2 in a star topology increases the expected request delay slightly while significantly decreasing the expected number of requests.

The bottom graph in Figure 7 shows the results from simulations, which concur with the analytical results in the top graph. For each integer value of C_2 from 0 to 100, twenty simulations are run, and the request delay and total number of requests is calculated for each simulation. Each simulation is represented by a jittered dot, and the line shows the average for each value of C_2. Thus, the graph shows that for C_2 set to 100, the average number of requests is 1.5 and the average request delay, as a multiple of the RTT, is 1.42. The minimum request delay of 1 comes from the fixed value of 2 for request parameter C_1.

Figure 8 shows the results from the chain topology discussed in Section 1. For a chain, with C_2 set to zero there will be exactly one request, with request delay $C_1/(2D)$. Increasing C_2 can increase both the expected request delay and the expected number of duplicates. The four lines in Figure 8 show the results for a chain topology with a failed edge 1, 2, 5, or 10 hops, respectively, from the source of the dropped packet. The individual simulations are shown by a dot only for the simulations with a failed edge one hop from the source. For each scenario C_2 ranges from 0 to 10 in increments of 1, and then from 10 to 100 in increments of 10. While increasing C_2 can increase the number of duplicates, the magnitude of the increase is quite small.

Figures 9 and 10 show the results for a range of tree topologies. Each line shows the results for a particular fixed scenario, as C_2 varies from 0 to 100. In all of the scenarios the session size is at least 100. We define the *density* of a tree as the fraction of nodes that are members of the multicast session. In each graph, the lines represent scenarios that differ only in the number of hops between the source and the failed edge. The four lines represent scenarios with failed edges that are one, two, three, or four hops, respectively, from the source of the dropped packet. For all of the topologies, the failed edge closest to the source gives the line with the worst-case number of duplicate requests. For this line, the individual simulations are each shown by a jittered dot. Note that the graphs do not necessarily show all of the dots.

As an example, the top graph in Figure 9 shows the results for trees of density 1. For each of the lines the average number of duplicates is minimized for C_2=0, and maximized for an intermediate value of C_2. In particular, for a failed edge adjacent to the source of the failed packet, C_2 set to 40 gives an average number of duplicates of 4.1. However, the only simulations in this section that give unacceptably large numbers of requests are those with small values for C_2 on stars or on trees with sparse sessions.

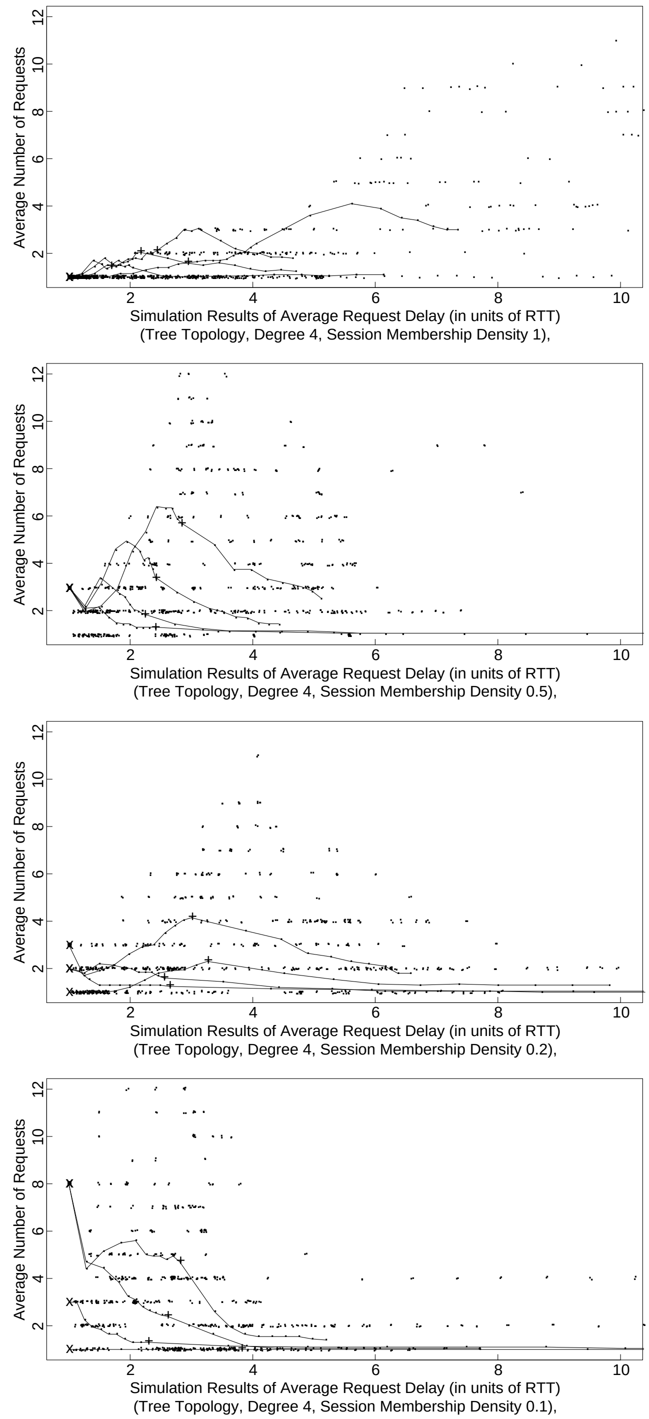

Figure 9: Tradeoff between delay and duplicates in dense tree topologies.

7 Extending the basic approach

7.1 Adaptive adjustment of random timer algorithms

The close connection of the loss recovery performance with the underlying topology of the network suggests that the

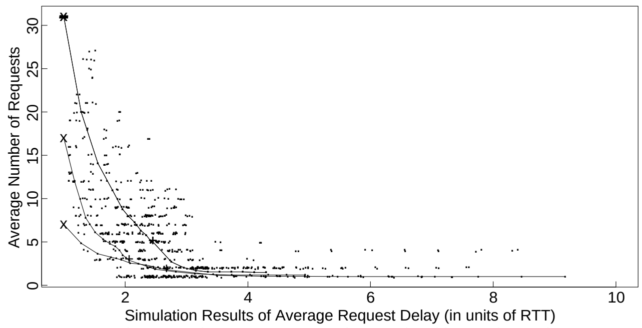

Figure 10: Tradeoff between delay and duplicates in a sparse tree topology.

timer parameters C_1, C_2, D_1, and D_2 be adjusted in response to the past behavior of the loss recovery algorithms. In this section we describe an adaptive algorithm that adjusts the timer parameters as a function of both the delay and of the number of duplicate requests and repairs in recent loss recovery exchanges.

For sparse sessions, which we expect to be the most common, there is a tradeoff between the delay and the number of duplicates; increasing C_2 decreases the expected number of duplicate requests but increases the expected request delay. However, the exact nature of the duplicate/delay curve depends on the topology and on the (possibly changing) failure scenario and session membership. Thus, for a sparse session the approach is to adjust C_2 dynamically, as a function of the past history of the request algorithms, to achieve the desired tradeoff between duplicates and delay.

The previous section shows that for dense trees where the failed edge is close to the source, increasing C_2 equally for all session members can increase both the delay and the number of duplicate requests. In this case, one strategy to minimize the number of duplicate requests is to rely on deterministic suppression, with members closest to the point of failure sending requests first. This deterministic suppression can be created in a dense tree if the request parameters C_1 and C_2 are reasonably small for at least one member close to the point of failure.

One mechanism for encouraging deterministic suppression in a dense tree is for members to reduce C_1 after they send a request. Members who frequently send requests are likely to also be members who are close to the point of failure, and in a dense tree, reducing C_1 for those members aids the deterministic suppression. In a star topology without deterministic suppression it does no harm to reduce C_1 for members who frequently send requests. Reducing C_1 in this fashion can actually help to break symmetry in a star, encouraging certain members to continue sending requests early.

A second mechanism for encouraging deterministic suppression in a dense tree is for members who have sent requests to reduce C_2 if the duplicate requests that they have received came from members significantly further from the source of the failed packet. This mechanism for requests requires that requests include the requestor's estimated distance from the original source of the requested packet. The corresponding mechanism for replies requires that replies include the replier's estimated distance from the source of the request.

```
After sending a request:
   decrease start of req. timer interval
Before each new request timer is set:
   if requests sent in prev. rounds, and any
   dup. requests were from further away:
      decrease request timer interval
   else if ave. dup. requests high:
      increase request timer interval
   else if ave. dup. requests low
   and ave. req. delay too high:
      decrease request timer interval
```

Figure 11: Dynamic adjustment algorithm for request timer interval.

Figure 11 gives the outline of a duplicates-based dynamic adjustment algorithm for adjusting the request timer parameters. A corresponding algorithm applies for adjusting the reply timer parameters. This adaptive algorithm combines a general adaption performed by all members when they set a request timer with more specific adaptions performed only by members who have recently sent requests. For the general adaption, if the average number of duplicate requests is too high, then the adaptive algorithm increases the request timer interval. Alternately, if the average number of duplicates is okay but the average delay in sending a request is too high, then the adaptive algorithm decreases the request timer interval. In this fashion the algorithm can adapt the timer parameters not only to fit the fixed underlying topology, but also to fit a changing session membership and pattern of congestion.

First we describe how a session member measures the average delay and number of duplicate requests in previous loss recovery rounds in which that member has been a participant. A *request period* begins when a member first detects a loss and sets a request timer, and ends only when that member begins a new request period. The variable *dup_req* keeps count of the number of duplicate requests received during one request period; these could be duplicates of the most recent request or of some previous request, but do not include requests for data for which that member never set a request timer. At the end of each request period, the member updates *ave_dup_req*, the average number of duplicate requests per request period, before resetting *dup_req* to zero. The average is computed as an exponential-weighted moving average,

$$ave\_dup\_req \leftarrow (1-\alpha)\ ave\_dup\_req + \alpha\ dup\_req,$$

with α=1/4 in our simulations. Thus, *ave_dup_req* gives the average number of duplicate requests for those request events for which that member has actually set a request timer.

When a request timer either expires or is reset for the first time, indicating that either this member or some other member has sent a request for that data, the member computes req_delay, the delay from the time the request timer was first set (following the detection of a loss) until a request was sent (as indicated by the time that the request timer either expired or was reset). The variable req_delay expresses this delay as a multiple of the roundtrip time to the source of the missing data. The member computes the average request delay, ave_req_delay.

In a similar fashion, a *repair period* begins when a member receives a request and sets a repair timer, and ends when a new repair period begins. In computing *dup_rep*, the number of duplicate repairs, the member considers only those repairs for which that member at some point set a repair timer. At the end of a repair period the member updates *ave_dup_rep*, the average number of duplicate repairs.

When a repair timer either expires or is cleared, indicating that this member or some other member sent a repair for that data, the member computes rep_delay, the delay from the time the repair timer was set (following the receipt of a request) until a repair was sent (as indicated by the time that the repair timer either expired or was cleared). As above, the variable rep_delay expresses this delay as a multiple of the roundtrip time to the source of the missing data. The member computes the average repair delay, ave_rep_delay.

Figure 12 gives the adaptive adjustment algorithm used in our simulator to adjust the request timer parameters C_1 and C_2. The adaptive algorithm is based on comparing the measurements *ave_dup_req* and *ave_req_delay* with AveDups and AveDelay, the target bounds for the average number of duplicates and the average delay. An identical adjustment algorithm is used to adapt the repair timer parameters D_1 and D_2, based on the measurements *ave_dup_rep* and *ave_rep_delay*. Figure 13 gives the initial values used in our simulations for the timer parameters. All four timer parameters are constrained to stay within the minimum and maximum values in Figure 13.

The numerical parameters in Figure 12 of 0.05, 0.1, and 0.5 were chosen somewhat arbitrarily. The adjustments of ± 0.05 and + 0.1 for C_1 are intended to be small, as the adjustment of C_1 is not the primary mechanism for controlling the number of duplicates. The adjustments of -0.1 and + 0.5 for C_2 are intended to be sufficiently small to minimize oscillations in the setting of the timer parameters. Sample trajectories of the loss recovery algorithms confirm that the variations from the random component of the timer algorithms dominate the behavior of the algorithms, minimizing the effect of oscillations.

```
After a request timer expires or is first
        reset:
   update ave_req_delay
After sending a request:
   C1 -=0.1
Before each new request timer is set:
   update ave_dup_req
   if closest_requestor on past requests:
      C2 -=0.1
   else if (ave_dup_req ≥ AveDups)):
      C1 +=0.1
      C2 +=0.5
   else if (ave_dup_req < AveDups-ε):
      if (ave_req_delay > AveDelay):
         C2 -=0.1
      if (ave_dup_req < 1/4):
         C1 -=0.05
   else C1 +=0.05
```

Figure 12: Dynamic adjustment algorithm for request timer parameters. In our simulations ϵ=0.1

```
Initial values:
   C1=2
   D1=log10 G
   C2=2
   D2=log10 G
Fixed parameters:
   MinC1=0.5; MaxC1=2
   MinC2=1; MaxC2=G
   MinD1=0.5; MaxD1=log10 G
   MinD2=1; MaxD2=G
   AveDups=1
   AveDelay=1
```

Figure 13: Parameters for adaptive algorithms

In our simulations we use a multiplicative factor of 3 rather than 2 for the request timer backoff described in Section 3.2. With a multiplicative factor of 2, and with an adaptive algorithm with small minimum values for C_1, a single node that experiences a packet loss could have its backed-off request timer expire before receiving the repair packet, resulting in an unnecessary duplicate request.

We have not attempted to devise an optimal adaptive algorithm for reducing some function of both delay and of the number of duplicates; such an optimal algorithm could involve rather complex decisions about whether to adjust mainly C_1 or C_2, possibly depending on such factors as that member's relative distance to the source of the lost packet. In a sparse tree, increasing C_2 reduces the number of duplicate

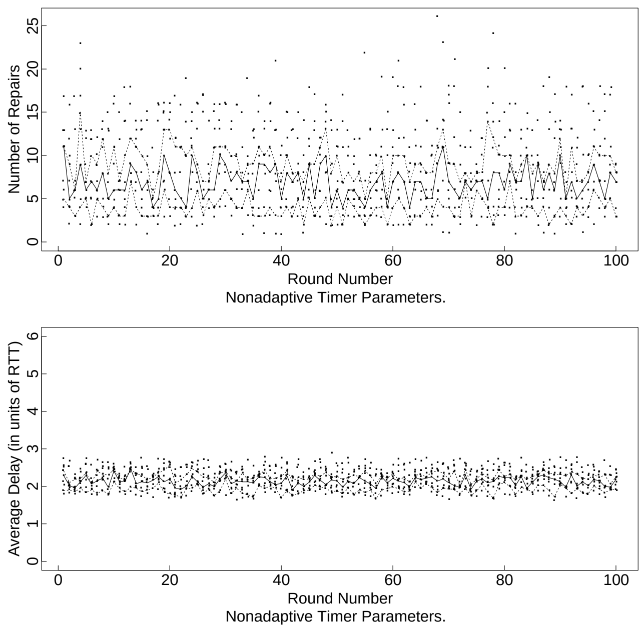

Figure 14: The non-adaptive algorithm.

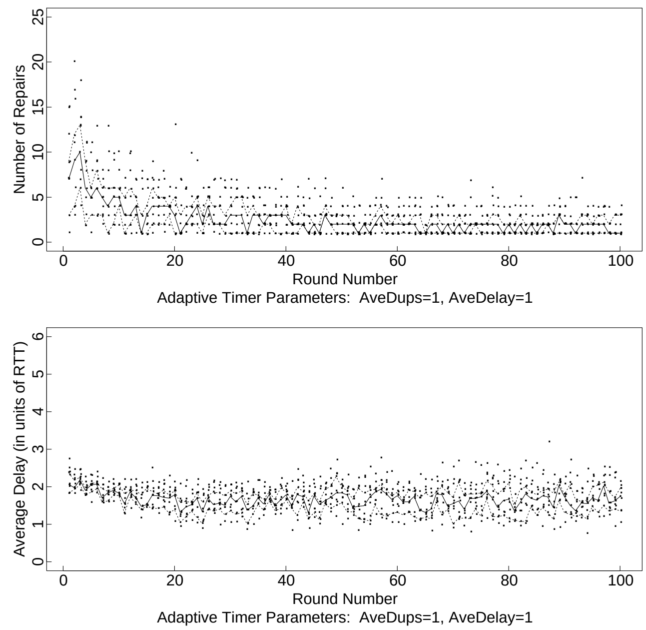

Figure 15: The adaptive algorithm.

requests; our adaptive algorithm relies largely on increases of C_2 to reduce duplicates. Our adaptive algorithm also decreases C_2 for members who have sent requests, if duplicate requests have come from members further from the source of the requested packet. (In our simulations "further from the source" is defined as "at a reported distance greater than 1.5 times the distance of the current member".) Our adaptive algorithm only decreases C_1 for members who have sent requests, or when the average number of duplicates is already small.

Figures 14 and 15 show simulations comparing adaptive and non-adaptive algorithms. The simulation set in Figure 14 uses fixed values for the timer parameters, and the one in Figure 15 uses the adaptive algorithm. From the simulation set in Figure 6, we chose a network topology, session membership, and drop scenario that resulted in a large number of duplicate requests with the non-adaptive algorithm. The network topology is a bounded-degree tree of 1000 nodes with degree 4 for interior nodes, and the multicast session consists of 50 members.

Each of the two figures shows ten runs of the simulation, with 100 loss recovery rounds in each run. For each round of a simulation, the same topology and loss scenario is used, but a new seed is used for the pseudo-random number generator to control the timer choices for the requests and repairs. In each round a packet from the source is dropped on the congested link, a second packet from the source is not dropped, and the loss recovery algorithms are run until all members have received the dropped packet. The x-axis of each graph shows the round number. For each figure, the top graph shows the number of requests in that rounds, and the bottom graph shows the loss recovery delay. Each round of each simulation is marked with a jittered dot, and a solid line shows the median from the ten simulations. The dotted lines show the upper and lower quartiles.

For the simulations in Figure 14 with fixed timer parameters, one round differs from another only in that each round uses a different set of random numbers for choosing the timers.

For the simulations with the adaptive algorithm in Figure 15, after each round of the simulation each session member uses the adaptive algorithms to adjust the timer parameters, based on the results from previous rounds. Figure 15 shows that for this scenario, the adaptive algorithms quickly reduce the average number of repairs, along with a small reduction in delay.

Figures 16 and 17 show the request and repair timer parameters for three 200-round executions of the simulations in Figure 15. For this scenario, the loss neighborhood consists of only two members, and the number of duplicate requests can be at most one. For each execution of the simulation, Figure 16 shows the request timer parameters C_1 and C_2 for both session members in the loss neighborhood. For each of the three simulations, a line marked "A" show the request parameters for the member closer to the point of failure, and a line marked "B" shows the request parameters for the member further away. For both nodes the parameter C_2 is slowly decreased to its minimum value, while C_1 is lower for the node closer to the point of failure.

Figure 17 shows the repair timer parameters D_1 and D_2 for two of the session members not in the loss neighborhood, the one closest to the point of failure (represented by the three

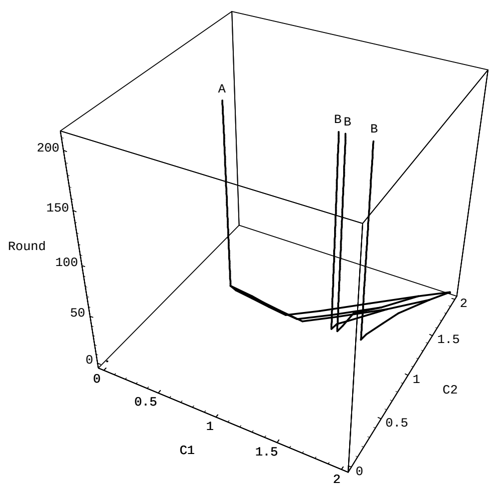

Figure 16: Request timer parameters for three executions of the simulation.

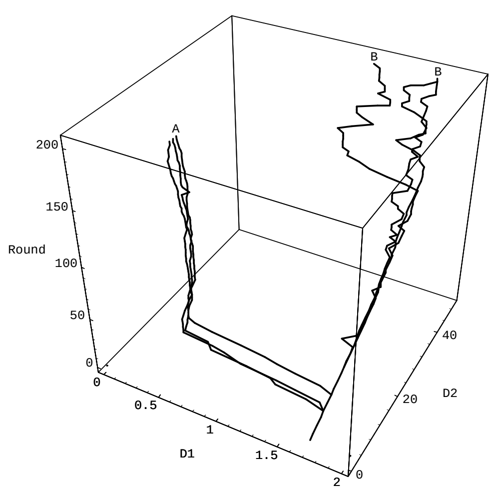

Figure 17: Repair timer parameters for three executions of the simulation.

lines marked "A"), and the other further away (represented by the three lines marked "B"). After the 100th round, for the member further from the point of failure the parameter D_2 has almost reached its maximum value of 50, and D_2 remains close to 50 for the remaining rounds. The initial rapid increase of D_2 results in a decrease in the number of duplicate repairs. At the same time, D_2 remains small for the member closest to the point of failure.

Figure 18 shows the results of the adaptive algorithm on the same set of scenarios as that in Figure 6. For each scenario (i.e., network topology, session membership, source member, and congested link) in Figure 18, the adaptive algorithm is run repeatedly for 40 loss recovery rounds, and Figure 18 shows the results from the 40th loss recovery round. Comparing Figures 6 and 18 shows that the adaptive algorithm is effective in controlling the number of duplicates over a range of scenarios.

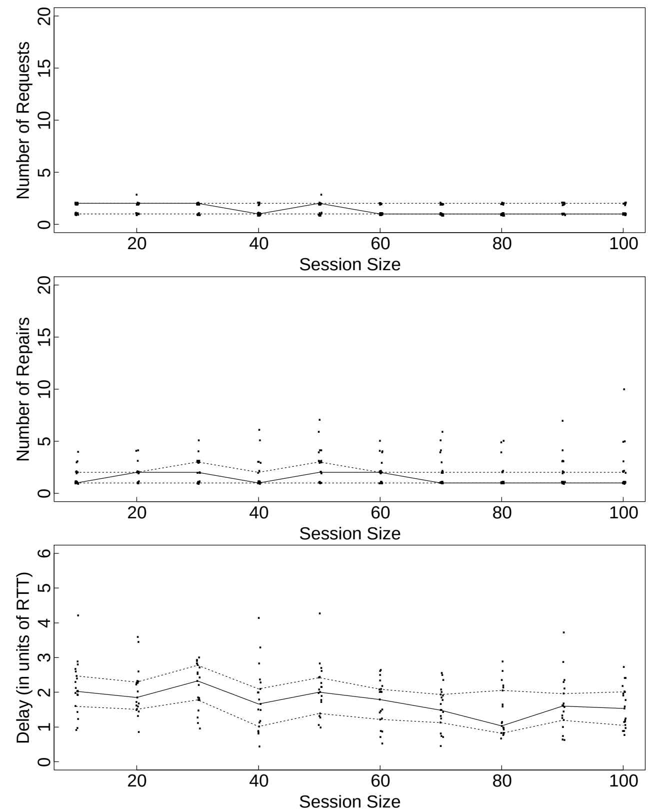

Figure 18: Adaptive algorithm on round 40, for a bounded-degree tree of 1000 nodes with degree 4 and a randomly picked congested link.

Simulations in [FJLMZ95] show that the adaptive algorithm works well in a wide range of conditions. These include scenarios where only one session member experiences the packet loss; where the congested link is chosen adjacent to the source of the packet to be dropped; and for a range of underlying topologies, including 5000-node trees, trees with interior nodes of degree 10; and connected graphs that are more dense that trees, with 1000 nodes and 1500 edges.

In actual multicast sessions, successive packet losses are not necessarily from the same source or on the same network link. Simulations in [FJLMZ95] show that in this case, the adaptive timer algorithms tune themselves to give good average performance for the range of packet drops encountered. Simulations in [FJLMZ95] show that, by choosing different values for AveDelay and AveDups, tradeoffs can be made between the relative importance of low delay and a low number of duplicates.

In the simulations in this section, none of the requests or repairs are themselves dropped. In more realistic scenarios where not only data messages but requests and repairs can be dropped at congested links as well, members have to rely on retransmit timer algorithms to retransmit requests and repairs as needed. Obviously, this will increase not only the delay, but also the number of duplicate requests and repairs in different parts of the network. The use of local recovery, described in the following section, would help to reduce the unnecessary use of bandwidth in the loss recovery algorithms.

7.2 Local recovery

With the global loss recovery algorithm described above, even if a packet is dropped on a link to a single member, both the request and the repair are multicast to the entire group. In cases where the neighborhood affected by the loss is small, the bandwidth costs of the loss recovery algorithm can be reduced if requests and repairs are multicast to a limited area. For example, studies of packet loss patterns in the current Mbone [YKT95] suggest that packet loss in multicast traffic is most likely to occur not in the "backbone" but in the "edges" of the multicast network. Scenarios that could benefit from local recovery include sessions with persistent losses to a small neighborhood of members and isolated late arrivals to a multicast session that ask for back history.

In this section we show that local recovery can be quite effective in reducing the unnecessary use of bandwidth. We are not at this stage proposing a specific set of mechanisms for implementing local recovery; there are still many open questions. For example, what algorithms should a member use to decide whether to use global or local scope for a specific request? How are these algorithms affected by application-specific relative priorities between minimizing delay and minimizing bandwidth? What are the best mechanisms to implement the "local-recovery neighborhoods" discussed below?

Local recovery assumes that the member sending the request has some information about the neighborhood of members sharing recent losses. However, end nodes should not know about network topology. We define a *loss neighborhood* as a set of members who are all experiencing the same set of losses. End nodes can learn about "loss neighborhoods" from information in session messages, without learning about the network topology. For each member, we call a loss a *local loss* if the number of members experiencing the loss is much smaller than the total number of members in the session. To help identify loss neighborhoods, session messages could report the names of the last few local losses. In addition, session messages could report the fraction of received repairs that are *redundant*, that is, those repairs received for known data, for which that member never set a request timer. A member could use local recovery when past losses have often been limited to members of a single loss neighborhood, and a request sent locally seems likely to reach some member capable of answering the request. If no repair is received before a backed-off request timer expires, then the second request can be sent with global scope.

One mechanism described in [FJLMZ95] for implementing local recovery is to set an appropriate "hop count" in the *time-to-live* (TTL) field of the IP header of multicast packets. However, simulations in [FJLMZ95] suggest that even assuming an optimal execution of the local recovery algorithms, the effectiveness of a simple hop-count-based local recovery in limiting the unnecessary use of bandwidth can depend heavily on the topology of the underlying network.

However, for a topology with naturally-defined *neighborhoods*, and some mechanism for restricting request and repair messages to these neighborhoods, local recovery algorithms can be quite effective. Simulations in this section illustrate local recovery based on *local-recovery neighborhoods*, where there is some mechanism that allows each member in the local-recovery neighborhood to send requests and repairs only to the members within that neighborhood. It is appropriate for a member to send a request limited to a local-recovery neighborhood when past experience indicates both the loss neighborhood, and some member capable on answering requests, are likely to be contained within the local-recovery neighborhood. Note that a network using pure hop-count-based local recovery, where each link counts as one hop and has a threshold of one, is not using local-recovery neighborhoods as defined above. In such a network, nearby members send local-recovery messages to slightly different, overlapping sets of nodes, rather than to the same neighborhood.

For TTL-based local recovery, local-recovery neighborhoods can be created by assigning high thresholds to links at the boundaries of a neighborhood. [FJLMZ95] discusses in more detail the mechanisms that would be needed for TTL-based local recovery neighborhoods. Local-recovery neighborhoods could also be defined by administrative scoping mechanisms incorporated in the multicast routing [J94c], by boundaries between hierarchies in the underlying hierarchical multicasting [TD95], or by separately-created multicast groups.

To introduce a network with local-recovery neighborhoods, we create the mesh-based network shown in Figure 19, with a central mesh and subtrees attached to each node in the mesh. We chose this topology because it roughly matches the topology of the Mbone, which can be considered as consisting of naturally-defined neighborhoods separated by transcontinental links and by backbone links within countries. For a 1000-node network, the mesh contains 19 nodes, all but one of the subtrees contain 53 nodes, and each subtree is a bounded-degree tree of degree four. We assume that each subtree is a separate local-recovery neighborhood. For a TTL-based local recovery scheme, for example, this could be accomplished by assigning high thresholds to the links within the mesh, and thresholds of one to the links within the subtrees.

Figure 20 shows that local recovery based on local-recovery neighborhoods can be quite effective. For the simulations in Figure 20, the local recovery options are very simple. Each member only has two choices when it sends a packet: to send the packet to the entire multicast group, or

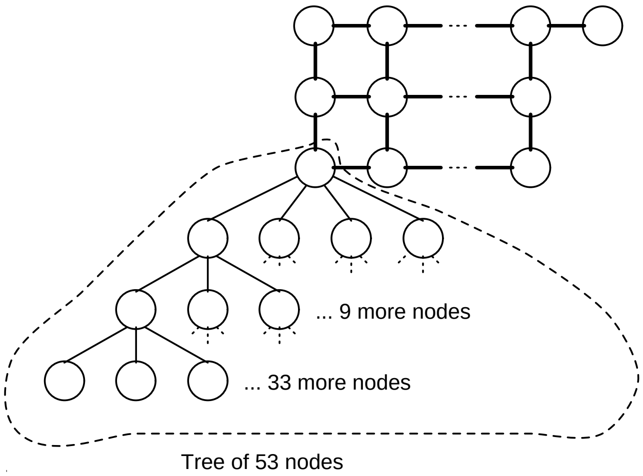

Figure 19: A mesh-based graph with subtrees of 53 nodes.

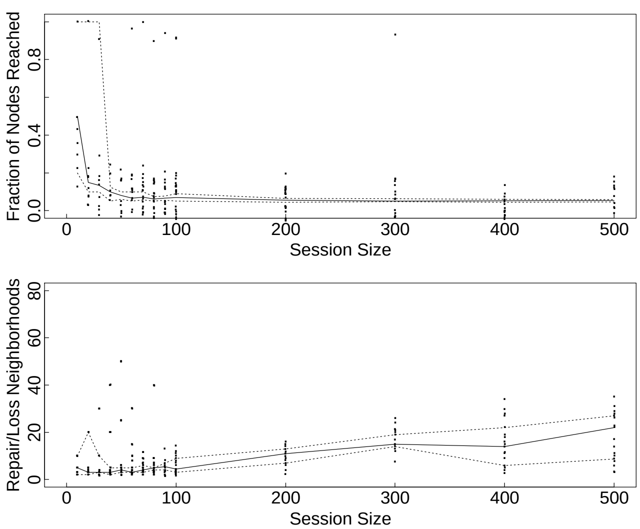

Figure 20: Local recovery in a mesh-based graph with 1000 nodes.

to send the packet only to those members within the same local-recovery group (i.e., subtree).

For each session size, twenty simulations are run, each with a different session membership and source and congested link for the dropped packet. We only consider scenarios where the loss neighborhood contains at most 10% of the session members. To explore the optimal possible performance of neighborhood-based local recovery algorithms in these simulations, when both the loss neighborhood and some member capable of answering a request are contained with the local-recovery neighborhood, the request is sent using local scope, and otherwise the request is sent using global scope.

In the top graph of Figure 20, the x-axis shows the session size, and the y-axis shows the fraction of session members reached by the repair. In the bottom graph, the y-axis shows the number of session members in the *repair neighborhood*, that is, the number of session members reached by the repair, as a multiple of the number of members in the loss neighborhood itself. The results of each simulation are represented by a jittered dot. The three lines indicate the first, second, and third quartiles. Figure 20 shows that neighborhood-based local recovery can be very effective in reducing the unnecessary use of bandwidth in the loss recovery events. With random choices for session members, congested links, and sources of the dropped packet, with the additional restriction that the loss neighborhood contain at most 10% of the session members, it is often the case for sessions with at least 20 members that both the loss neighborhood and some member capable of answering a request will be contained within a single local-recovery area.

8 Application-specific and general aspects of reliable multicast

Section 2 discussed some of the underlying assumptions in the design of reliable multicast for wb. In this section we explore some of the ways that the reliable multicast framework described in this paper could be used and modified to meet the needs of other applications for reliable multicast.

A fundamental concept in our reliable multicast algorithm is a *multicast group*, i.e. a set of hosts that (1) can be reached by a group address without being identified individually first, and (2) share the same application data and thus can help each other with loss recovery. This group concept is also appropriate for applications such as routing protocol updates and DNS updates, as well as for the group distribution of stock quotes, Usenet news, or WWW-based mass media.

Let's take the Border Gateway Protocol (BGP) as an example. The Internet is viewed as a set of arbitrarily connected autonomous systems (AS) that are connected through border gateways that speak BGP to exchange routing information. One AS may have multiple BGP speakers, and all BGP speakers representing the same AS must give a consistent image of the AS to the outside, i.e. they must maintain consistent routing information. In the current implementation, this consistency is achieved by each BGP router opening a TCP connection to each other BGP router to deliver routing updates reliably. There are several problems with this approach. First, achieving multicast delivery by multiple one-to-one connections bears a high cost. Second, for an AS with N BGP routers, one has to manually configure the $(N-1)$ TCP connections for each of the N routers, and repeat again whenever a change occurs. Both of these problems could be solved by applying our reliable multicast algorithm, perhaps with some minor adjustments to the data persistence model.

Our reliable multicast framework could easily be adapted for the distribution of such delay-insensitive material as Usenet news. Different applications have different tradeoffs between minimizing delay and minimizing the number of duplicate requests or repairs. For an interactive application such as wb, close attention must be paid to minimizing delay. For reliably distributing Usenet news, on the other hand, minimizing bandwidth would be more important than minimizing delay. Again some minor tuning to our request and repair timer algorithms may make our work readily applicable to the news distribution.

As a third example, we could consider applying the basic approach in this work to data caching and replication for Web pages. Like wb, all objects in the Web have a globally unique identifier. With HTTP, all requests for a specific object are handled by the original source, even though in many cases, especially for "hot" objects, a copy may be found within the neighborhood of a requester. As distributed Web caches are implemented, our reliable multicast framework could be used to reliably distribute updates to the caches. As another possibility, when a user makes a request to a remote object, the request could be multicast to a cache group. By using our timer algorithms, the cache in the cache group closest to the requester would be most likely to send a reply.

We believe that the approach to reliable multicast described in this paper could be useful to a wide range of applications based on multicast groups. Even for applications that may require partial or total data ordering, the reliable multicast framework described in this paper could be used to reliably deliver the data to all group members, and a partial or total ordering protocol could be built on top that is specifically tailored to the ordering needs of that application. As [CS93] has shown, for applications with ordering requirements, preserving the ordering of messages as they appear in the network is often an expensive and inadequate substitute for preserving the "semantic ordering" of the messages appropriate for the application.

9 Related work on reliable multicast

The literature is rich with architectures for reliable multicast [MTC]. Due to space limitations, we will not describe the details of each solution. Instead, we focus on the different goals and definitions of reliability in the various architectures, and the implications of these differences for the scalability, robustness, handling of dynamic group membership, and overhead of the algorithms.

The Chang and Maxemchuk protocol [CM84] is one of the pioneer works in reliable multicast protocols. It is basically a centralized scheme that provides totally ordered delivery of data to all group members. All the members are ordered in a logical ring, with one of them being the master, called the token site. The token site is moved around the ring after each data transmission. Sources multicast new data to the group, and the token site is responsible for acknowledging (by multicast) the new data with a timestamp, as well as retransmitting (through unicast) all missing packets upon requests from individual receivers. The order of data reception at all the sites is determined by the timestamp in the ACK. Each ACK also serves to pass the token to the next member in the ring. By shifting the token site among all the members, with a requirement that a site can become the token site only if it has received all the acknowledged data, it is assured that after shifting the token site through all the N members in the group, everyone will have received all the data that is at least N smaller than the current timestamp value.

Because the token site is responsible for all the acknowledgments and retransmissions, it becomes the bottleneck point whenever losses occur. The scheme also requires reformation of the ring whenever a membership change occurs. Therefore it does not scale well with the size of the group.

RMP (Reliable Multicast Protocol) [WKM95], based on the Chang and Maxemchuk algorithm, provides an atomic, totally ordered, reliable multicast service that runs on top of IP Multicasting. RMP provides added QoS parameters in each data transfer and better handling of membership changes. The most recent version of RMP uses a modified SRM Request/Repair policy along with a sliding window flow control scheme based on TCP [MWC95].

The reliable multicast protocol for ordered delivery described in [KTHB89] is similar to, but simpler than, the Chang and Maxemchuk protocol. Basically, all data is first unicast to a master site, called a sequencer, which then multicasts the data to the group. Therefore the sequencer provides a global ordering of all the data in time; it is also responsible for retransmitting, by unicast, all the missing data upon requests. The sequencer site does not move unless it fails, in which case a new sequencer is elected. To avoid keeping all the data forever, the sequencer keeps track of the receiving state of all the members to determine the highest sequence number that has been correctly received by all the members.

MTP (Multicasting Transport Protocol) [AFM92] is again a centralized scheme for totally ordered multicast delivery. A master site is responsible for granting membership and tokens for data transmission; each host must obtain a token from the master first before multicasting data to the group, thus the total order of data packets is maintained. A window size defines the number of packets that can be multicast into the group in a single heartbeat and a retention size defines the period (in heartbeats) to maintain all client data for retransmission. NACKs are unicast to the data source which then multicasts the retransmission to whole group.

Compared to the above cited works, the Trans and Total protocols described in [MMA90] are closer in spirit to our work. These protocols assume that all the members in a multicast group are attached to one broadcast LAN. Each host keeps an acknowledgment list which contains identifiers of both positive and negative ACKs. Whenever a host sends a data packet, it attaches its acknowledgment list to the packet, as a way to synchronize the state with all other members in the group. Because the single LAN limits data transmissions from all hosts to one packet at a time, partial and total ordering of data delivery can be readily derived from data and acknowledgment sequences.

Log-based Receiver-reliable Multicast (LBRM) [HSC95] was designed to support Distributed Interactive Simulation (DIS). The receiver-based reliability is provided by primary and secondary logging servers. Receivers request retransmissions from the secondary logging servers, which requests retransmissions from the primary logging server. Both the source and the secondary logging servers use either deterministic or probabilistic requests to select between unicast and multicast retransmissions.

LBRM uses a variable heartbeat scheme sends heartbeat messages (e.g., session messages) more frequently immediately after a data transmission. In an environment when the basic transmission rate is low, this variable heartbeat enables receivers to detect losses sooner, with no penalty in terms of the total number of heartbeat messages transmitted. While the variable heartbeat scheme would not be appropriate for an application such as wb, where the original congestion could itself result from many senders sending data at the same time, the variable heartbeat scheme could be quite useful for an application with a natural limit on the worst-case number of concurrent senders, and would be easily implementable in SRM.

Perhaps the most well-known work on reliable multicast is the ISIS distributed programming system developed at Cornell University [BSS91, ISIS]. ISIS provides causal ordering and, if desired, total ordering of messages *on top of* a reliable multicast delivery protocol. Therefore the ISIS work is to some extent orthogonal to the work described in this paper, and further confirms our notion that a partial or total ordering, when desired, can always be added on top of a reliable multicast delivery system. The reliable multicast delivery in existing ISIS implementations is achieved by multiple unicast connections using a windowed acknowledgment protocol similar to TCP [B93]. Horus, the successor to ISIS, can optionally run on top of IP multicast.

10 Future work

10.1 Future work on local recovery

Section 7.2 has shown that local recovery based on local-recovery neighborhoods can be quite effective in limiting the unnecessary use of bandwidth in loss recovery events. While [FJLMZ95] discusses some of the issues in implementing TTL-based local recovery, there are many open questions about which mechanisms should be used to define local-recovery neighborhoods, how individual members should determine whether to send requests with local or global scope, etc.

In many topologies, the effectiveness of local recovery could be improved by adding members to the multicast group in strategic locations. For example, consider the known stable topologies discussed in [HSC95], where losses are expected to occur mainly on the tail circuits, rather than in the backbone or in the LANs, and the design priority is to keep unnecessary traffic off of the tail circuits. The addition of a session member (i.e., cache) on a node near the local end of the tail circuit, coupled with a local-recovery neighborhood defined to include all members on that end of the tail circuit, would allow local recovery to continue for losses on the local area without adding any unnecessary traffic to the tail circuit itself. And for losses on the tail circuit itself, defining a larger local recovery area that included the local end of the tail circuit and some member (or cache) on the backbone end of the tail circuit would enable local recovery for losses on the tail circuit to continue without adding unnecessary traffic to the other tail circuits.

10.2 Future work on congestion control

SRM's basic framework for congestion control assumes that the members of the multicast session have an estimate of the *available bandwidth* for the session, and constrain the data transmitted to be within this estimated bandwidth. This framework raises several somewhat separate issues, such as how members determine this available bandwidth; how to detect congestion or avoid potential congestion; and given available bandwidth, which piece of data a member should send first.

Multicast congestion control is a relatively new area for research. For unicast traffic, there is a single path from source to receiver, with an automatic feedback loop provided by returning acknowledgment packets. In contrast, in a multicast group there could be several sources, and the various communication paths from an active source to the members of the multicast group can have a range of bandwidth, propagation delay, and competing congestion. In this case, how does one define and detect congestion? Do the sources respond to congestion by slowing down their transmission rate, do the receivers respond to congestion by unsubscribing from (possibly layered) multicast groups, or should there be some combination of these two approaches? With multicast traffic, there are application-specific policy decisions about whether or not to tune the congestion control procedures to the needs of the worst-case receiver; these questions do not arise with unicast transmissions. In this section we assume a scalable application such as wb that is not necessarily tuned to the needs of the worst-case receiver.

It is possible for congestion to be detected collectively by the members in a session, for example through observations of packet losses or of the data reception rate. As in RTP, session messages can be used to exchange information about observed performance.

If the bandwidth along different paths in the multicast group differs substantially, then members behind small pipes that detect severe congestion can unsubscribe from the group, or perhaps from one of several multicast groups associated with that session. An approach under investigation for the video tool vic [MV95] is to divide the total data transmission into several substreams, with each being sent to a separate multicast group. Members that detect congestion would unsubscribe from higher-bandwidth groups. If this framework is used for reliable multicast, then reliable delivery should be provided separately within each group.

While considerable research has been done on layering techniques for video, layering techniques are application-specific, and layering for wb data remains an area for further research. As a simple example of layering for wb data, a low-bandwidth multicast group could be limited to text-based data, and a higher-bandwidth multicast group could be used for graphics or for side-discussions. Other possibilities would be to encode embedded images using progressive JPEG or some other layered scheme, or to tradeoff free-hand drawing resolution for rate (i.e., you could send 50 points/sec on a high rate channel and 1 point/sec on a low-rate channel).

As another approach to assure a certain amount of bandwidth for a session, receivers could reserve resources where such network services were available; an example of such services are the predictive and controlled delay services currently being developed for the Internet [BCS94]. We believe that it should be the choice of individual members whether to reserve resources or to rely on best effort service for a session; the use of services other than best-effort should not be a mandated requirement for all members of a multicast group. Thus, resource reservation could complement other congestion control mechanisms of the multicast session. The current implementation of wb essentially uses a static assumption of the available bandwidth, backed up by the informal, consensus-based "admissions-control procedure" of the current Mbone.

Independent from how we estimate or obtain the available bandwidth, individual members must constrain the aggregate data transmission by the session to be under this available bandwidth. Under this constraint, however, one can apply application-specific policies to determine the transmission order of packets. In wb, for example, priority in data transmission goes to loss recovery for the current "page", then to new data, and last to loss recovery for previous pages.

11 Conclusions

This paper described in detail SRM, a scalable reliable multicast algorithm that was first developed to support wb. We have discussed the basic design principles as well as extensions of the basic algorithm that make it more robust for a wide range of network topologies.

Many applications need or desire support for reliable multicast. Experience with the wb design shows, however, that individual applications may have widely different requirements of multicast reliability. Instead of designing a generic reliable multicast protocol to meet the most stringent requirements, this work has resulted in a simple, robust, and scalable reliable multicast algorithm that meets a minimal reliability definition of delivering all data to all group members, leaving more advanced functionalities, whenever needed, to be handled by individual applications.

The work described in this paper is based on the fundamental principles of application level framing (ALF), multicast grouping, and the adaptivity and robustness in the TCP/IP architecture design. Although our current protocol implementation is specifically tailored to wb, the protocol framework is generally applicable to a wide variety of other applications.

12 Acknowledgments

This work benefited from discussions with Dave Clark and with the End-to-End Task Force about general issues of sender-based vs. receiver-based protocols. We would also like to thank Peter Danzig for discussions about reliable multicasting and web-caching.

References

[AFM92] Armstrong, S., Freier, A., and Marzullo, K., "Multicast Transport Protocol", *Request for Comments (RFC) 1301*, Feb. 1992.

[B93] Birman, K., "The Process Group Approach to Reliable Distributed Computing", *Communications of the ACM*, Dec. 1993.

[BSS91] Birman, K., Schiper, A., and Stephenson, P., "Lightweight Casual and Atomic Group Multicast", *ACM Transactions on Computer Systems*, Vol.9, No. 3, pp. 272-314, Aug. 1991.

[BCS94] B. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", *Request for Comments (RFC) 1633*, IETF, June 1994.

[CM84] Chang, J., and Maxemchuk, N., "Reliable Broadcast Protocols", *ACM Transactions on Computer Systems*, Vol.2, No. 3, pp. 251-275, Aug. 1984.

[CS93] Cheriton, D., and Skeen, D., "Understanding the Limitations of Causally and Totally Ordered Communication", *Proceedings of the 14th Symposium on Operating System Principles*, ACM, December 1993.

[CT90] Clark, D., and Tennenhouse, D., "Architectural Considerations for a New Generation of Protocols", *Proceedings of ACM SIGCOMM '90*, Sept. 1990, pp. 201-208.

[CLZ87] Clark, D., Lambert, M., and Zhang, L., "NETBLT: A High Throughput Transport Protocol", *Proceedings of ACM SIGCOMM '87*, pp. 353-359, Aug. 1987.

[D91] Deering, S., "Multicast Routing in a Datagram Internetwork", PhD thesis, Stanford University, Palo Alto, California, Dec. 1991.

[FJLMZ95] Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, Extended Report", LBNL Technical Report, URL ftp://ftp.ee.lbl.gov/papers/wb.tech.ps.Z, Sept. 1995.

[HSC95] Holbrook, H., Singhal, S., and Cheriton, D., "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation", *Proceedings of ACM SIGCOMM '95*, August 1995.

[ISIS] ISIS and Horus WWW page, URL http://www.cs.cornell.edu/Info/Projects/ISIS/ISIS.html.

[J92] Jacobson, V., "A Portable, Public Domain Network 'Whiteboard' ", Xerox PARC, viewgraphs, April 28, 1992.

[J94] Jacobson, V., "A Privacy and Security Architecture for Lightweight Sessions", Sante Fe, NM, Sept. 94.

[J94c] Jacobson, V., "Administratively Scoped IP Multicast", viewgraphs, 30th IETF, Toronto, Canada, July 25, 1994. URL ftp://ftp.ee.lbl.gov/talks/adminscope.ps.Z.

[J94a] Jacobson, V., "Multimedia Conferencing on the Internet", Tutorial 4, SIGCOMM 1994, Aug. 1994.

[KTHB89] Kaashoek, M., Tannenbaum, A., Hummel, and Bal, "An Efficient Reliable Broadcast Protocol", *Operating Systems Review*, Oct., 1989.

[M92] McCanne, S., "A Distributed Whiteboard for Network Conferencing", May 1992, UC Berkeley CS 268 Computer Networks term project.

[MJ95] McCanne, S., and Jacobson, V., "vic: A Flexible Framework for Packet Video", to appear in *ACM Multimedia 1995*.

[MV95] McCanne, S., and Vetterli, M., "Joint Source/Channel Coding for Multicast Packet Video", *Proceedings of the IEEE International Conference on Image Processing*, Washington, D.C., October, 1995.

[MMA90] Melliar-Smith, P., Moser, L., and Agrawala, V., "Broadcast Protocols for Distributed Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1 No. 1, Jan. 1990, pp. 17-25.

[M84] Mills, D.L., "Network Time Protocol (Version 3)", RFC (Request For Comments) 1305, March 1992.

[MWC95] Montgomery, T., Whetten, B., and Callahan, J., "The Reliable Multicast Protocol Specification Flow Control and NACK Policy", July 1995, URL ftp://research.ivv.nasa.gov/pub/doc/RMP/RMPflow.txt.

[MTC] Multicast Transport Protocols WWW page, URL http://hill.lut.ac.uk/DS-Archive/MTP.html.

[Pa85] Palmer, E., *Graphical Evolution: An Introduction to the Theory of Random Graphs*, John Wiley & Sons, 1985.

[SCFJ94] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications", *Internet Draft* draft-ietf-avt-rtp-06.txt, work in progress, Nov. 1994.

[PTK94] Pingali, S., Towsley, D., and Kurose, J., "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols", *SIGMETRICS '94*.

[TD95] Thyagarajan, A., and Deering, S., "Hierarchical Distance-Vector Multicast Routing for the Mbone", *ACM SIGCOMM 95*, pp. 60-65.

[TS94] Thyagarajan, A., and Deering, S., IP Multicast release 3.3, Aug. 1994, available from ftp://parcftp.xerox.com/pub/ net-research /ipmulti3.3-sunos413x.tar.Z.

[WKM95] Whetten, B., Kaplan, S., Montgomery, T., "A High Performance Totally Ordered Multicast Protocol", submitted to *INFOCOM '95*.

[YKT95] Yajnik, M., Kurose, J., and Towsley, D., "Packet Loss Correlation in the MBone Multicast Network: Experimental Measurements and Markov Chain Models", submitted to *INFOCOM '95*.